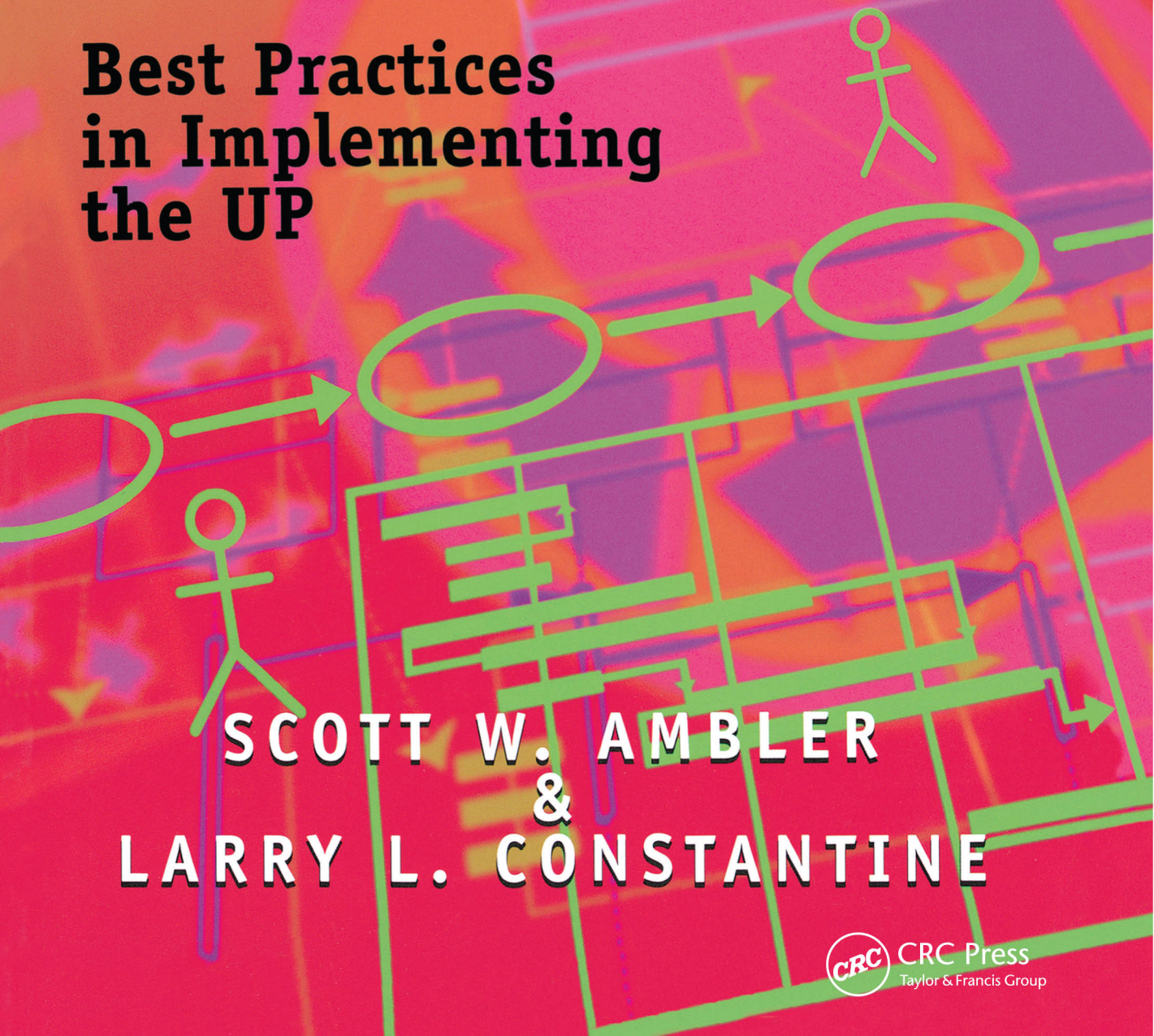
The Unified Process INCEPTION PHASE

Best Practices in Implementing the UP



SCOTT W. AMBLER
&
LARRY L. CONSTANTINE

CRC Press
Taylor & Francis Group

The Unified Process Inception Phase

Best Practices in Implementing the UP

Scott W. Ambler and Larry L. Constantine, Compiling Editors

Masters collection from

SOFTWARE Development

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

First issued in hardback 2017

ISBN 13: 978-1-138-41226-2 (hbk)
ISBN 13: 978-1-929629-10-7 (pbk)

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Cover art design: Robert Ward and John Freeman

To my aunts and uncles:
Bruce, Shirley, June, Peter, Viola, and Ross.

— Scott Ambler

Table of Contents

Foreword

There was a time when one or two talented programmers with a vision could create useful, even groundbreaking, software applications on their own. Today, though, most significant applications demand the resources of a team of professionals with diverse skills to create and refine a substantial body of code and supporting documents. The most effective teams thoughtfully apply sensible software development processes that suit the nature of their project, its context and constraints, and the organization's culture.

One of the well-established, contemporary software development processes is the Unified Process from Rational Corporation. You can buy books that describe the Unified Process in great detail. Such books tell you what to do, but they provide scant guidance about how to do it practically, effectively, and efficiently. Editors Scott Ambler and Larry Constantine close the gap in this book and the others in the series. Scott and Larry bring a vast background of software experience, insight, and perspective to the table. In this book, they collect the even broader experience, insight, and perspective contributed by dozens of authors to more than ten years of *Software Development* and *Computer Language* magazine. The forty-one articles, together with Scott and Larry's additional commentary, provide a wealth of guidance on how to tackle the first phase of the enhanced lifecycle for the Unified Process, the Inception phase. Few of these articles describe projects that specifically followed the Unified Process. However, Scott and Larry have selected key papers that address important recurring themes in software development — best practices — and aligned them with the Unified Process phases. Every software engineer and project manager should know of these best practices and understand how to apply them to whatever process their project uses.

The Inception phase is perhaps the most critical of all. It deals with the "fuzzy front end" of the project, in which the team lays the foundation for success — or for failure. If you don't have a good understanding of your customers' functional, usability, and quality needs, it doesn't matter how well you execute the later phases of Elaboration, Construction, Transition,

and Production. If you begin with unclear business objectives, ill-understood user requirements, unrealistic plans and schedules, or unappreciated risks, your project is born with one foot already in the grave.

The Unified Process recognizes that key process workflows, including requirements, implementation, test, and project management, span multiple lifecycle phases. This book therefore includes articles that provide specific recommendations on how to perform the critical activities in each workflow during the Inception phase. Scott Ambler and Larry Constantine also recognize that no single process or methodology can supply a formulaic solution to the diverse challenges and situations a software development organization will encounter. Nonetheless, this book will go a long way toward helping you apply the Unified Process to your projects. And even if you don't care about the Unified Process, the articles provide a solid foundation of current thinking on software engineering best practices.

Karl E. Wiegers
Principal Consultant at Process Impact

Preface

A wealth of knowledge has been published on how to be successful at developing software in *Software Development* magazine and in its original incarnation, *Computer Language*. The people who have written for the magazine include many of the industry's best known experts: Karl Wiegers, Steve McConnell, Ellen Gottesdiener, Jim Highsmith, Warren Keuffel, and Lucy Lockwood, to name a few. In short, the leading minds of the information industry have shared their wisdom with us over the years in the pages of this venerable magazine.

Lately, there has been an increased focus on improving the software process within most organizations. This is in part due to the Year 2000 (Y2K) debacle, to the significant failure rate of large-scale software projects, and to the growing realization that following a mature software process is a key determinant in the success of a software project. In the mid-1990s, Rational Corporation began acquiring and merging with other tool companies. As they consolidated the companies, they consolidated the processes supported by the tools of the merging companies. The objective of the consolidation was to arrive at a single development approach. They named the new approach the Unified Process. Is it possible to automate the entire software process? Does Rational have a complete toolset even if it is? We're not so sure. Luckily, other people were defining software processes too, so we have alternate views of how things should work. This includes the OPEN Consortium's OPEN process, the process patterns of the Object-Oriented Software Process (OOSP), and Extreme Programming (XP). These alternate views can be used to drive a more robust view of the Unified Process, resulting in an enhanced lifecycle that more accurately reflects the real-world needs of your organization. Believing that the collected wisdom contained in *Software Development* over the years could be used to flesh-out the Unified Process — truly unifying the best practices in our industry — we undertook this book series.

Why is a software process important? Step back for a minute. Pretend you want to have a house built and you ask two contractors for bids. The first one tells you that, using a new housing technology, he can build a house for you in two weeks if he starts first thing tomorrow, and it will cost you only $100,000. This contractor has some top-notch carpenters and plumbers that have used this technology to build a garden shed in the past, and they're willing

to work day and night for you to meet this deadline. The second one tells you that she needs to discuss what type of house you would like built, and then, once she's confident that she understands your needs, she'll put together a set of drawings within a week for your review and feedback. This initial phase will cost you $10,000, and, once you decide what you want built, she can then put together a detailed plan and cost schedule for the rest of the work.

Which contractor are you more comfortable with — the one that wants to start building or the one that wants to first understand what needs to be built, model it, plan it, then build it? Obviously, the second contractor has a greater chance of understanding your needs, the first step for successfully delivering a house that meets them. Now assume that you're having software built — something that is several orders of magnitude more complex and typically far more expensive than a house, and assume once again that you have two contractors wanting to take these exact same approaches. Which contractor are you more comfortable with? We hope the answer is still the second one; the one with a sensible process. Unfortunately, practice shows that most of the time, organizations appear to choose the approach of the first contractor; that of hacking. Of course, practice also shows that our industry experiences upwards of 85% failure rate on large-scale, mission-critical systems. (In this case, a failure is defined as a project that has significantly overrun its cost estimates or has been cancelled outright.) Perhaps the two phenomena are related.

In reality, the situation is even worse than this. You're likely trying to build a house and all the contractors that you have available to you only have experience building garden sheds. Even worse, they've only worked in tropical parts of the world and have never had to deal with the implications of frost, yet you live in the backwoods of Canada. Furthermore, the various levels of Canadian government enforce a variety of regulations that the contractors are not familiar with — regulations that are constantly changing. Once again, the haphazard approach of the first contractor is likely to get one into trouble.

The Inception Phase

In the enhanced lifecycle for the Unified Process, the Inception phase is the first of five phases — Inception, Elaboration, Construction, Transition, and Production — that a release of software experiences throughout its complete lifecycle. The primary goal of the Inception phase is to set a firm foundation for your project. To accomplish this, you will need to:

- justify both the system itself and your approach to developing/obtaining the system,
- describe the initial requirements for your system,
- determine the scope of your system,
- identify the people, organizations, and external systems that will interact with your system,
- develop an initial risk assessment, schedule, and estimate for your system, and
- develop an initial tailoring of the Unified Process to meet your exact needs.

When you step back and think about it, the most important thing that you can do is ensure that your system, and your approach to it, is justified (i.e., that you have a business case). If the project doesn't make sense — either from an economic, technical, or operational point of view — then you shouldn't continue. Seven out of eight large-scale projects fail. Without a firm foundation, an architecture that will work, a realistic project plan, and a committed team of professionals, your project is very likely going to be one of the seven failures.

This book collects articles written by industry luminaries that describe best practices in these areas. One goal of this book, and of the entire series, is to provide proven alternative approaches to the techniques encompassed by the Unified Process. Another goal is to fill in some of the gaps in the Unified Process. Because the Unified Process is a development process, not a software process that covers development and the operations and support of software once in production, it inevitably misses or shortchanges some of the concepts that are most important for software professionals. Fortunately, the writers in *Software Development* have taken a much broader view of process scope and have filled in many of these holes for us.

About This Series

This book series comprises four volumes: one for the Inception phase, one for the Elaboration phase, one for the Construction phase, and a fourth one for the Transition and Production phases. Each book stands on its own, but for a complete picture of the entire software process you need the entire series. The articles presented span the complete process without duplication among the volumes.

Overall organization of this book series.

Workflow/Topic	Inception Phase (volume 1)	Elaboration Phase (volume 2)	Construction Phase (volume 3)	Transition and Production Phases (volume 4)
Business Modeling	X	X		
Requirements	X	X		
Analysis and Design		X	X	
Implementation			X	
Test	X	X	X	X
Deployment				X
Operations and Support				X
Configuration and Change Management			X	
Project Management	X	X	X	X
Environment	X			X
Infrastructure Management		X	X	X

It has been a challenge selecting the material for inclusion in this compilation. With such a wealth of material to choose from and only a limited number of pages in which to work, narrowing the choices was not always easy. If time and space would have allowed, each of these books might have been twice as long. In narrowing our selections, we believe the articles that remain are truly the *crème de la crème*. Furthermore, to increase the focus of the material in each book, we have limited the number of workflows that each one covers. Yes,

most workflows are pertinent to each phase; but as the previous table indicates, each book covers a subset to provide you with a broader range of material for those workflows.

About the Editors

Scott Ambler

An avid reader of *Computer Language* and then *Software Development* for years, I started writing for the magazine in 1995 and eventually became the object columnist in 1997. I started developing software in the early 1980s, writing code in languages such as Fortran and Basic, and later in the mid-1980s in Turing (don't ask), C, Prolog, and Lisp. In the late 1980s, I realized that there was more to life than programming and started picking up skills in user interface design, data modeling, process modeling, and testing while I programmed in COBOL and a couple of fourth-generation languages for IBM mainframes. Disillusioned with structured/procedural techniques, in 1990, I discovered objects and readily jumped into Smalltalk development, then into C++ development, then back to Smalltalk. Having worked at several organizations in mentoring and architectural roles, I decided to combine that experience and apply my skills gained as a teaching assistant at the University of Toronto and get into professional training in the mid-1990s. I quickly learned several things. First, that although I liked delivering training courses (and still do so today), I didn't want to do it full time. Second, and of greater importance, I learned how to communicate complex concepts in an easy-to-understand manner, such as how to develop object-oriented software. This led to my first two books, *The Object Primer* (Cambridge University Press, 1995), now in its second edition (2000), and *Building Object Applications That Work* (Cambridge University Press, 1997/1998), which describe the fundamentals of object technology from a developer's point of view. I then decided to follow up with two books that describe the Object-Oriented Software Process (OOSP) in *Process Patterns* (Cambridge University Press, 1998) and *More Process Patterns* (Cambridge University Press, 1999), focusing on the hard-won experiences that I gained working for one of Canada's leading object technology consulting firms. Since then I've helped several organizations, large and small, new and established, in a variety of industries to improve their internal software processes. My latest writing endeavors include this book series as well as co-authoring *The Elements of Java Style* (Cambridge University Press, 2000). I am now President of Ronin International (`www.ronin-intl.com`), a Denver-based process and software architecture consulting firm, and a freelance writer with my own web site, `www.ambysoft.com`, where I post a variety of white papers. I think I've found my niche.

Larry Constantine

My association with *Software Development* and its forerunner, *Computer Language*, has been both long and fruitful, and my association with software development and computer language goes back even further. From my first Fortran program back in the dark ages of computing, I have been keenly interested in figuring out how to do things better and to help others do them better — interests that soon led me beyond technology into management and process issues, as well as the essential matter of the usability of the products we design and build. Throughout my nearly 40 years in the field, I have continued to criss-cross that river that too often divides the people side from the technology side. In my view, success in soft-

ware development hinges on an understanding and a mastery of material from both sides of this divide, and this has been reflected in my writing for the magazine and elsewhere. That work now spans over 150 articles and papers and 14 books, including, now, this collaborative compilation with Scott Ambler. With Scott's concurrence, some of my own columns and articles in the magazine have been included in these volumes. Others appear in *The Peopleware Papers* (Prentice Hall, 2000), which reprints in its entirety the contents of my long-running "Peopleware" column, and in *Managing Chaos: The Expert Edge in Software Development* (Addison-Wesley, 2000), which incorporates the best from the popular Software Development Management Forum that appears at the back of every issue. In recent years, my professional interests have been particularly focused on increasing the usability of software, which has led to the development of usage-centered design and to the book with Lucy Lockwood, *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design* (Addison-Wesley, 1999). The magazine honored us by giving that book the Jolt Product Excellence Award for best book of 1999. Of late, it seems I cross oceans even more often than rivers, because, although I live in the United States, I also teach at the University of Technology, Sydney, Australia, where I am an Adjunct Professor of Computing Sciences. Despite the title, I teach a mix of management and design topics. I am also a working trainer, designer, and consultant helping clients around the world build software that is easier to use. With Lucy Lockwood, I founded Constantine & Lockwood, Ltd. (`www.forUse.com`), where I am Director of Research and Development and am currently working on the integration of usage-centered design with the Unified Process and Unified Modeling Language, among other things.

Acknowledgments

We'd like to thank the following people for their insightful comments during the development of this book: Susan Ambler, John Nalbone, Mark Peterson, Neil Pitman, Doug Smith, Art Staden, and Robert J. White Jr.

1

Chapter 1

Introduction

What is a software process? A software process is a set of project phases, stages, methods, techniques, and practices that people employ to develop and maintain software and its associated artifacts (plans, documents, models, code, test cases, manuals, etc.). Not only do you need a software process, you need one that is proven to work in practice — a software process tailored to meet your exact needs.

Why do you need a software process? An effective software process will enable your organization to increase its productivity when developing software. First, by understanding the fundamentals of how software is developed, you can make intelligent decisions, such as knowing to stay away from SnakeOil v2.0 — the wonder tool that claims to automate fundamental portions of the software process. Yes, tools are important, but they must support and accommodate the chosen process, and they should not introduce too much overhead. Second, it enables you to standardize your efforts, promoting reuse, repeatability, and consistency between project teams. Third, it provides an opportunity for you to introduce industry best practices such as code inspections, configuration management, change control, and architectural modeling to your software organization. Fourth, a defined software process establishes a baseline approach for greater consistency and future enhancements.

An effective software process will also improve your organization's maintenance and support efforts — also referred to as production efforts — in several ways. First, it should define how to manage change and appropriately allocate maintenance changes to future releases of your software, streamlining your change process. Second, it should define both how to transition software into operations and support smoothly and how the operations and support efforts are actually performed. Without effective operations and support processes, your software will quickly become shelfware.

An effective software process considers the needs of both development and production.

Why adopt an existing software process, or improve your existing process using new techniques? The reality is that software is growing more and more complex, and without an effective way to develop and maintain that software, the chance of achieving the required levels of quality decreases. Not only is software getting more complex, you're also being asked to create more software simultaneously. Most organizations have several software projects in development at one time and have many moare than that in production — projects that need to be managed effectively. Furthermore, the nature of the software that we're building is also changing — from the simple batch systems of the 1970s for which structured techniques were geared toward, to the interactive, international, user-friendly, 7/24, high-transaction, high-availability online systems that object-oriented and component-based techniques are aimed. And while you're doing that, you're asked to increase the quality of the systems that you're delivering, and to reuse as much as possible so that you can work faster for less money. A tall order — one that is nearly impossible to fill if you can't organize and manage your staff effectively. A software process provides the basis to do just that.

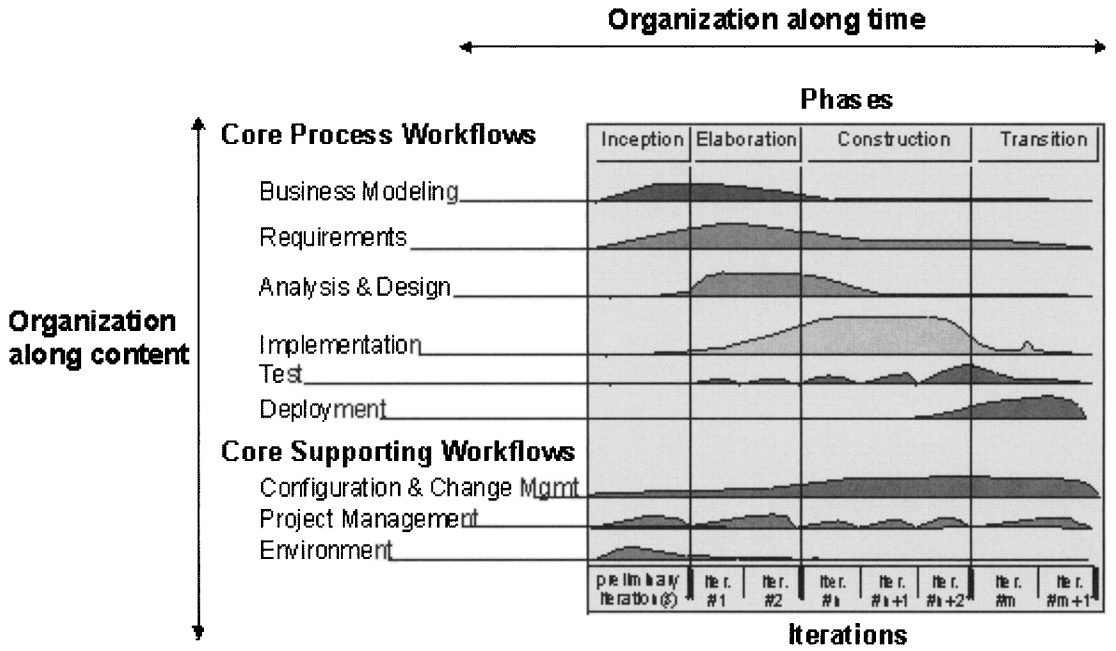
Software is becoming more complex, not less.

1.1 The Unified Process

The Unified Process is the latest endeavor of Rational Corporation (Kruchten, 2000), the same people who introduced what has become the industry-standard modeling notation, the Unified Modeling Language (UML). The heart of the Unified Process is the Objectory Process, one of several products and services that Rational acquired when they merged with Ivar Jacobson's Objectory organization several years ago. Rational enhanced Objectory with their own processes (and those of other tool companies that they have either purchased or partnered with) to form the initial version (5.0) of the Unified Process officially released in December of 1998.

Figure 1.1 presents the initial lifecycle of the Unified Process comprised of four serial phases and nine core workflows. Along the bottom of the diagram, you can see that any given development cycle through the Unified Process should be organized into iterations. The basic concept is that your team works through appropriate workflows in an iterative manner so that at the end of each iteration, you produce an internal executable that can be worked with by your user community. This reduces the risk of your project by improving communication between you and your customers. Another risk-reduction technique built into the Unified Process is the concept that you should make a "go/no-go" decision at the end of each phase — if a project is going to fail, then you want to stop it as early as possible. Granted, the important decision points are actually at the end of the Inception and Elaboration phases (by the time you've hit the end of the Construction phase, it's usually too late to cancel). This is an important concept in an industry with upward of an 80%–90% failure rate on large-scale, mission-critical projects (Jones, 1996).

Figure 1.1 The initial lifecycle of the Unified Process.



The Inception phase, the topic of this volume, is where you define the project scope and the business case for the system. The initial use cases for your software are identified and the key ones are described briefly. Use cases are the industry standard technique for defining the functional requirements for systems. They provide significant productivity improvements over traditional requirement documents because they focus on what adds value to users as opposed to product features. Basic project management documents are started during the Inception phase, including the initial risk assessment, the estimate, and the project schedule. As you would expect, key tasks during this phase include business modeling and requirements engineering, as well as the initial definition of your environment, including tool selection and process tailoring.

You define the project scope and the business case during the Inception phase.

The Elaboration phase focuses on detailed analysis of the problem domain and the definition of an architectural foundation for your project. Because use cases aren't sufficient for defining all requirements, a deliverable called a *supplementary specification* is defined which describes all non-functional requirements for your system. A detailed project plan for the Construction Phase is also developed during this phase based on the initial management documents started in the Inception phase.

You define the architectural foundation for your system during the Elaboration phase.

The Construction phase is where the detailed design for your application is developed as well as the corresponding source code. The goal of this phase is to produce the software and supporting documentation to be transitioned to your user base. A common mistake that project teams make is to focus primarily on this phase, often to their detriment because organizations typically do not invest sufficient resources in the previous two phases and therefore lack the foundation from which to successfully develop software that meets the needs of their users. During the Inception and Elaboration phases, you invest the resources necessary to understand the problem and solution domains. During the Construction phase, there should be very little "surprises," such as significantly changed requirements or new architectural approaches — your goal is to build the system.

You finalize the system to be deployed during the Construction phase.

The purpose of the Transition phase is to deliver the system to your user community. There is often a beta release of the software to your users — typically called a pilot release within most businesses — in which a small group of users work with the system before it is released to the general community. Major defects are identified and potentially acted upon during this phase. Finally, an assessment is made regarding the success of your efforts to determine whether another development cycle/increment is needed to further enhance the system. It is during this time that your non-functional requirements, including technical constraints such as performance considerations, become paramount. You will focus on activities such as load testing, installation testing, and system testing — all activities that validate whether or not your system fulfills its non-functional requirements. As you will see in Chapter 3, there is far more to developing requirements than simply writing use cases.

You deliver the system during the Transition phase.

The Unified Process has several strengths. First, it is based on sound software engineering principles such as taking an iterative, requirements-driven, and architecture-based approach to development in which software is released incrementally. Second, it provides several mechanisms, such as a working prototype at the end of each iteration and the "go/no-go" decision point at the end of each phase, which provides management visibility into the development process. Third, Rational has made, and continues to make, a significant investment in their Rational Unified Process (RUP) product (`http://www.rational.com/products/rup`), an HTML-based description of the Unified Process that your organization can tailor to meet its exact needs. In fact, the reality is that you must tailor it to meet *your* needs because at 3,000+ HTML pages, it comprises far more activities than any one project, or organization, requires. Pick and choose from the RUP the activities that apply, then enhance them with the best practices described in this book series and other sources to tailor a process that will be effective for your team. Accepting the RUP right out of the box is naïve at best — at worst, it is very likely a path to failure.

Attempting to use the Unified Process out of the box is a recipe for woe and strife. — Neil Pitman

The Unified Process also suffers from several weaknesses. First, it is only a development process. The initial version of the Unified Process does not cover the entire software process. As you can see in Figure 1.1, it is obviously missing the concept of operating and supporting your software once it has been released into production. Second, the Unified Process does not explicitly support multi-project infrastructure development efforts such as organization/enterprise-wide architectural modeling, missing opportunities for large-scale reuse within your organization. Third, the iterative nature of the lifecycle is foreign to many experienced developers, making acceptance of it more difficult, and the rendering of the lifecycle in Figure 1.1 certainly does not help this issue.

The software industry has a capacity for almost infinite self-delusion. — Capers Jones

In *The Unified Process Elaboration Phase* (Ambler & Constantine, 2000a), the second volume in this series, we show in detail that you can easily enhance the Unified Process to meet the needs of real-world development. We argue that you need to start with the requirements for a process — a good start at which is the Capability Maturity Model (CMM). Second, you should look at the competition — in this case, the OPEN Process (Graham, Henderson-Sellers, and Younessi, 1997; `http://www.open.org.au`) and the process patterns of the Object-Oriented Software Process (Ambler 1998, Ambler 1999), and see which features you can reuse from those processes. Figure 1.2 depicts the contract-driven lifecycle for the OPEN process and Figure 1.3 depicts the lifecycle of the Object-Oriented Software Process (OOSP), comprised of a collection of process patterns[1]. Finally, you should formulate an enhanced lifecycle based on what you've learned and support that lifecycle with proven best practices.

The Unified Process is a good start, but likely needs to be tailored and enhanced to meet the specific needs of your organization.

1. A *process pattern* is a collection of general techniques, actions, and/or tasks (activities) that solve a specific software process problem taking the relevant forces/factors into account. Just like design patterns describe proven solutions to common software design problems, process patterns present proven solutions to common software process patterns. More information regarding process patterns can be found at the Process Patterns Resource Page, `http://www.ambysoft.com/processPatternsPage.html`.

Figure 1.2 The OPEN Contract-Driven lifecycle.

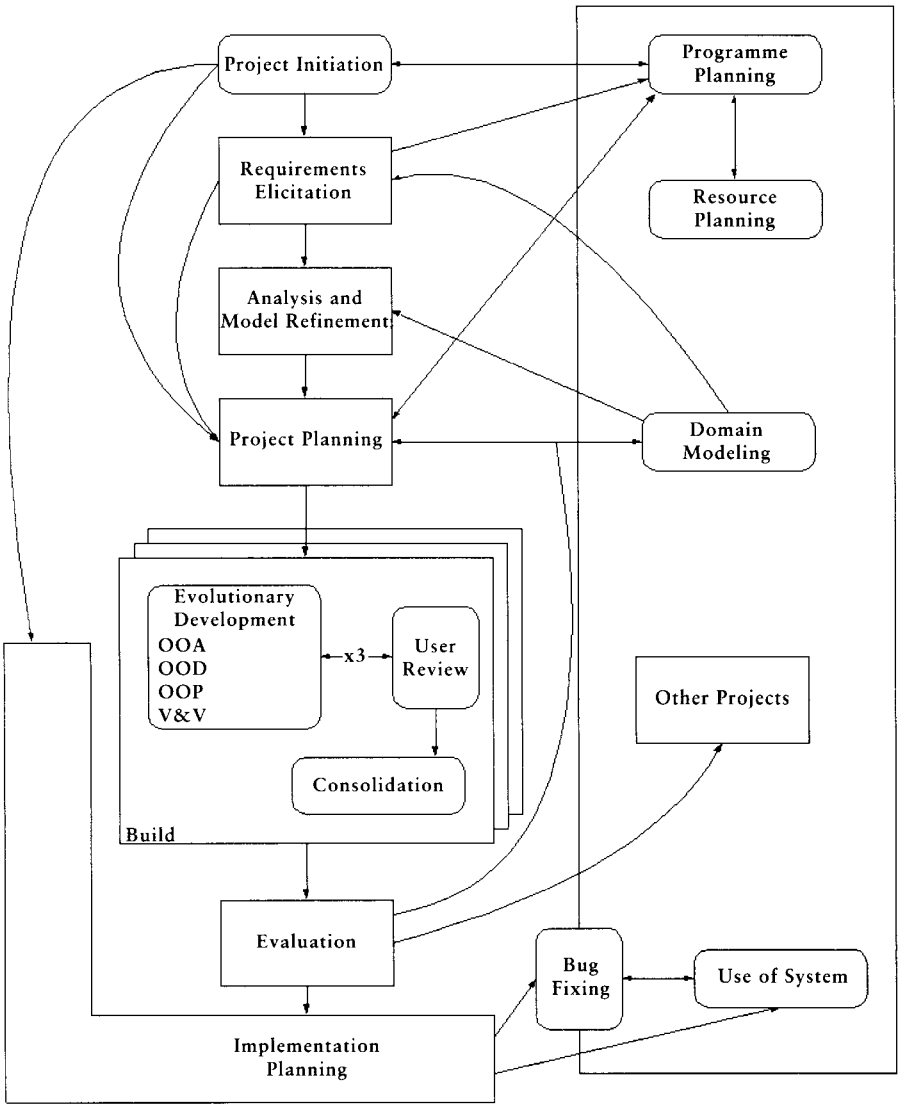
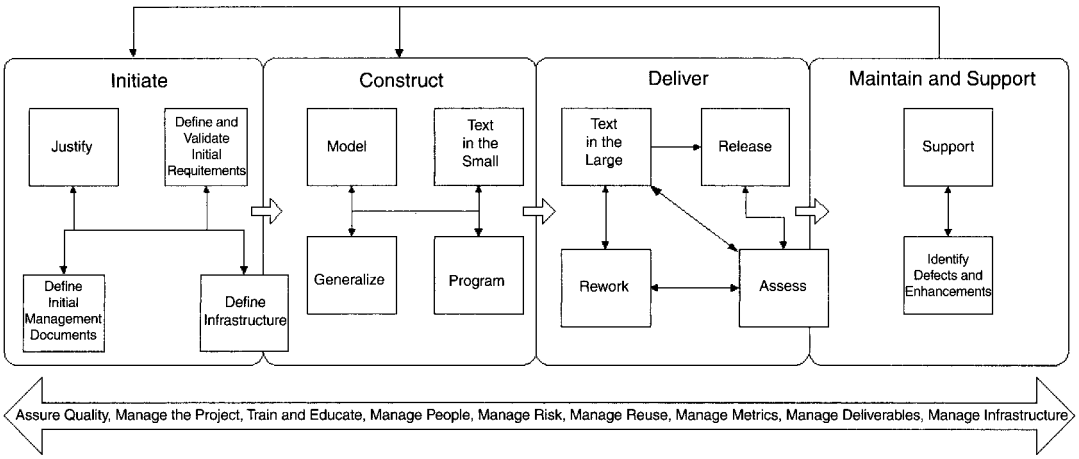


Figure 1.3 The Object-Oriented Software Process (OOSP) lifecycle.



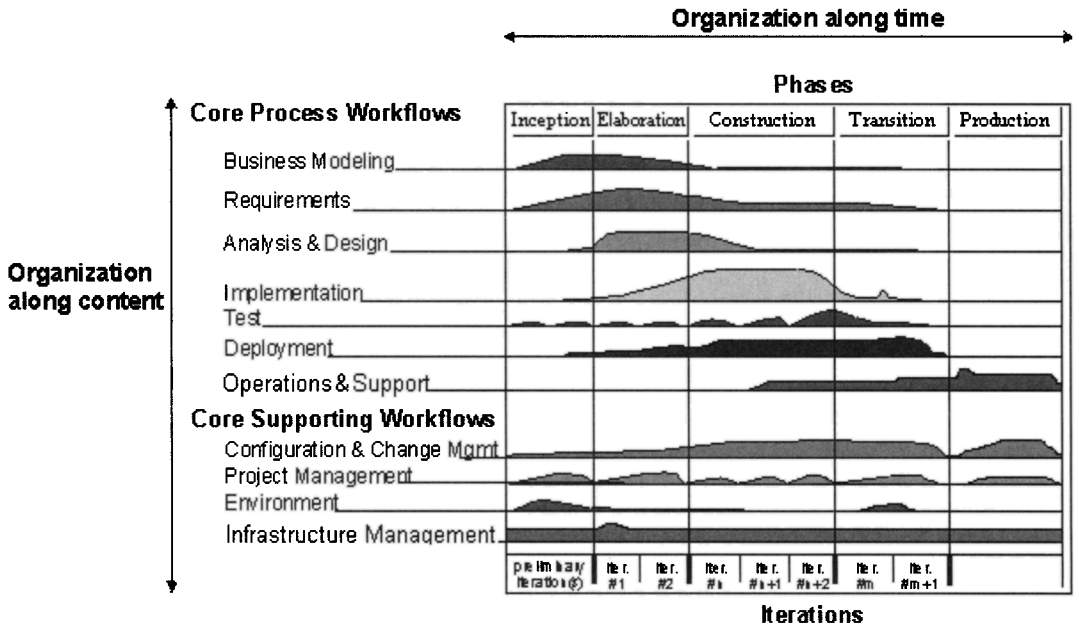
"Serial in the large, iterative in the small, delivering incremental releases over time."

1.2 The Enhanced Lifecycle for the Unified Process

You've seen overviews of the requirements for a mature software process and the two competing visions for a software process. Knowing this, how do you complete the Unified Process? Well, the first thing to do is to redefine the scope of the Unified Process to include the entire software process, not just the development process. This implies that software processes for operations, support, and maintenance efforts need to be added. Second, to be sufficient for today's organizations, the Unified Process also needs to support the management of a portfolio of projects — something the OPEN Process has called "programme management" and the OOSP has called "infrastructure management." These first two steps result in the enhanced version of the lifecycle depicted in Figure 1.4. Finally, the Unified Process needs to be fleshed out with proven best practices; in this case, found in articles published in *Software Development*.

The enhanced lifecycle includes a fifth phase, Production, representing the portion of the software lifecycle after a version of a system has been deployed. Because, on average, software spends 80% of its lifetime in production, the Production phase is a required feature of a realistic software process. Explicitly including a Production phase also enhances the 20% of the lifecycle that is spent in development because it makes it clear to developers that they need to take production issues into account and it provides greater motivation to work towards a common architecture across projects. As the name implies, its purpose is to keep your software in production until it is either replaced with an updated version — from a minor release such as a bug fix to a major new release — or it is retired and removed from production. Note that there are no iterations during this phase (or there is only one iteration depending on how you wish to look at it) because this phase applies to the lifetime of a single release of your software. To develop and deploy a new release of your software, you need to run through the four development phases again.

Figure 1.4 The enhanced lifecycle for the Unified Process.



The Production phase encompasses the post-deployment portion of the lifecycle.

Figure 1.4 also shows that there are two new workflows: a core workflow called *Operations & Support* and a supporting workflow called *Infrastructure Management*. The purpose of the Operations & Support workflow is exactly as the name implies: to operate and support your software. Operations and support are both complex endeavors, endeavors that need processes defined for them. This workflow, as well as all the others, span several phases. During the Construction phase, you will need to develop operations and support plans, documents, and training manuals. During the Transition phase, you will continue to develop these artifacts, reworking them based on the results of testing, and you will train your operations and support staff to effectively work with your software. Finally, during the Production phas,e your operations staff will keep your software running, performing necessary backups and batch jobs as needed, and your support staff will interact with your user community in working with your software. This workflow basically encompasses portions of the OOSP's Release stage and Support stage as well as the OPEN Process's *Implementation Planning* and *Use of System* activities. In the Internet economy, where 24/7 operations is the norm, you quickly discover that high quality and high availability is crucial to success — you need an Operations and Support workflow.

The Operations and Support workflow is needed to ensure high quality and high availability of your software.

The Infrastructure Management workflow focuses on the activities required to develop, evolve, and support your organization's infrastructure artifacts such as your organization/enterprise-wide models, your software processes, standards, guidelines, and your reusable artifacts. Your software portfolio management efforts are also performed in this workflow. Infrastructure Management occurs during all phases; the blip during the Elaboration phase represents architectural support efforts to ensure that a project's architecture appropriately reflects your organization's overall architecture. This includes infrastructure modeling activities such as the development of an enterprise requirements/business model, a domain architecture model, and a technical architecture model. These three core models form your infrastructure models that describe your organization's long-term software goals and shared/reusable infrastructure. The processes followed by your Software Engineering Process Group (SEPG) — responsible for supporting and evolving your software processes, standards, and guidelines — are also included in this workflow. Your reuse processes are included as well because practice shows that to be effective, reuse management needs to be a cross-project endeavor. For you to achieve economies of scale developing software, increase the consistency and quality of the software that you develop, and to increase reuse between projects, you need to manage your common infrastructure effectively. You need the Infrastructure Management workflow.

Infrastructure Management supports your cross-project/programme-level activities such as reuse management and organization/enterprise-wide architecture.

If you compare the enhanced lifecycle of Figure 1.4 with the initial lifecycle of Figure 1.1, you will notice that several of the existing workflows have also been updated. First, the Test workflow has been expanded to include activity during the Inception phase. You develop your initial, high-level requirements during this phase — requirements that you can validate using techniques such as walkthroughs, inspections, and scenario testing. Two of the underlying philosophies of the OOSP are that (a) you should test often and early and, (b) that if something is worth developing, then it is worth testing. Therefore, testing should be moved forward in the lifecycle. Also, the Test workflow needs to be enhanced with the techniques of the OOSP's *Test in the Small* and *Test in the Large* stages.

Test early and test often. If it is worth creating, it is worth testing.

The second modification is to the Deployment workflow — extending it into the Inception and Elaboration phases. This modification reflects the fact that deployment, at least of business applications, is a daunting task. Data conversion efforts of legacy data sources are often a project in their own right — a task that requires significant planning, analysis, and work to accomplish. Furthermore, our belief is that deployment modeling should be part of the Deployment workflow — not the Analysis & Design workflow as it is currently — due to the

fact that deployment modeling and deployment planning go hand-in-hand. Deployment planning can, and should, start as early as the Inception phase and continue into the Elaboration and Construction phases in parallel with deployment modeling.

Deployment is complex and planning often must start early in development to be successful.

The Environment workflow has been updated to include the work necessary to define the Production environment — work that would typically occur during the Transition phase. (You could easily do this work earlier if you wish, but the Transition phase often proves the best time for this effort). The existing Environment workflow processes effectively remain the same — the only difference being that they now need to expand their scope from being focused simply on a development environment to also include operations and support environments. Your operations and support staff need their own processes, standards, guidelines, and tools — the same as your developers. Therefore, you may have some tailoring, developing, or purchasing to perform to reflect this need.

The Configuration & Change Management workflow is extended into the new Production phase to include the change control processes needed to assess the impact of a proposed change to your deployed software and to allocate that change to a future release of your system. This change control process is often more formal than what you do during development due to the increased effort required to update and re-release existing software.

Change control management will occur during the Production phase.

Similarly, the Project Management workflow is also extended into the new Production phase to include the processes needed to manage your software once it has been released. It is light on metrics management activities and subcontractor management, a CMM level 2 key process area, needed by of any organization that outsources portions of its development activities or hires consultants and contractors. People management issues, including training and education as well as career management, are barely covered by the Unified Process because those issues were scoped out of it. There is far more to project management than the technical tasks of creating and evolving project plans. You also need to manage your staff and mediate the interactions between them and other people.

There is far more to project management than planning, estimating, and scheduling.

1.3 The Goals of the Inception Phase

During the Inception phase, your project team will focus on understanding the initial requirements, determining scope, and organizing the project. To understand the initial requirements, you will likely perform business modeling and essential modeling activities (Constantine and Lockwood, 1999). Essential models are intended to capture the essence of problems through technology-free, idealized, and abstract descriptions. Your resulting design models are more

flexible, leaving more options open and accommodating changes more readily in technology. Essential models are more robust than concrete representations, simply because they are more likely to remain valid in the face of both changing requirements and changes in the technology of implementation. Essential models of usage highlight purpose, what it is that users are trying to accomplish, and why they are doing it. In short, essential models are ideal artifacts to capture the requirements for your system.

Your project's scope can be determined through negotiation with project stakeholders as to the applicability of the results of your business and requirements modeling efforts. The optional development of a candidate architecture enables you to determine both the technical feasibility of your project and its scope because you may find that some aspects of your system are better left out of the current release. Significant project management effort also occurs during the Inception phase where you'll organize your future work, including the development of schedules, estimates, plans, and risk assessments.

The focus of the Inception phase is to define and come to agreement with respect to the high-level requirements, vision, and scope of your project, as well as to justify the project and obtain resources to continue work on it. As you work towards these goals, you will create and/or evolve a wide variety of artifacts, such as:

- A vision document
- An initial requirements model (10–20% complete)
- An initial project glossary
- A business case
- An initial domain model (optional)
- An initial business model (optional)
- A development case describing your project's tailored software process (optional)
- An architectural prototype (optional)

The phase is concluded with the Lifecycle Objective (LCO) milestone (Kruchten, 2000). To pass this milestone, you must achieve:

- a consensus between project stakeholders as to the project's scope and resource requirements,
- an initial understanding of the overall, high-level requirements for the system,
- a justification for your system that includes economic, technological, and operational issues,
- a credible, coarse-grained schedule for your entire project,
- a credible, fine-grained schedule for the initial iterations of the Elaboration phase,
- a credible, risk assessment and resource estimate/plan for your project,
- a credible initial tailoring of your software process,
- a comparison of your actual vs. planned expenditures to date for your project, and
- the development of an initial architectural prototype for your system (optional).

A beginning is the time for taking the most delicate care that the balances are correct. — Maud'Dib

1.4 How Work Generally Proceeds During the Inception Phase

A fundamental precept of the Unified Process is that work proceeds in an iterative manner throughout the activities of the various workflows. However, at the beginning of each iteration, you will spend more time in requirements-oriented activities and towards the end of each iteration, your focus will be on test-oriented activities. As a result, to make this book easier to follow, the chapters are organized in the general order by which you would proceed through a single iteration of the Inception phase. As Figure 1.4 indicates, the workflows applicable during the Inception phase are:

- Business Modeling (Chapter 2)
- Requirements (Chapter 3)
- Analysis & Design (covered in Vols. 2 and 3, *The Unified Process Elaboration Phase* and *The Unified Process Construction Phase*, respectively)
- Implementation (covered in Vols. 2 and 3, *The Unified Process Elaboration Phase* and *The Unified Process Construction Phase*, respectively)
- Test (Chapter 4)
- Deployment (covered in Vol. 4, *The Unified Process Transition Phase*)
- Operations and Support (covered in Vol. 4, *The Unified Process Transition Phase*
- Configuration & Change Management (covered in Vol. 3, *The Unified Process Construction Phase*)
- Project Management (Chapter 5)
- Environment (Chapter 6)
- Infrastructure Management (covered in Vols. 2 and 3, *The Unified Process Elaboration Phase* and *The Unified Process Construction Phase*, respectively)

1.4.1 The Business Modeling Workflow

The purpose of the Business Modeling workflow, described in detail in Chapter 2, is to model the business context of your system. During the Inception phase, the focus of the Business Modeling workflow is to:

Identify the context of your system. A context model shows how your system fits into its overall environment. This model will depict the key organizational units that will work with your system, perhaps the marketing and accounting departments, and the external systems that it will interact with. By developing the context model, you come to an understanding of the structure and culture of the organization and the external business environment your system will exist in and support.

Identify the basis for a common understanding of the system and its context with stakeholders. Your goal is to begin working towards a common understanding of what your project team will deliver to its stakeholders as well as a common understanding of the environment in which you will work. Stakeholders include your direct users, senior management, user management, your project team, your organization's architecture team, and potentially even your operations and support management. Without this common understanding,

your project will likely be plagued with politics and infighting and could be cancelled prematurely if senior management loses faith in it.

Model the business. When you are modeling a business, you want to understand its goals, strengths, weaknesses, and the opportunities and challenges that it faces in the market place. A business model is multifaceted — potentially including a process model, a use-case model, and a conceptual object model. Your project's business model should reflect your enterprise requirements model. Enterprise modeling is a key aspect of the Infrastructure Management workflow which is covered in detail in *The Unified Process Elaboration Phase* (Ambler & Constantine, 2000a) and *The Unified Process Construction Phase* (Ambler & Constantine, 2000b). Business process models show how things get done, as opposed to a use-case model that shows what should be done and can be depicted using UML activity diagrams. Conceptual object models show the major business entities, their responsibilities, and their inter-relationships, and should be depicted using UML class diagrams. Your business model should include a glossary of key terms and important technical terms optional that your project stakeholders need to understand. The business model is important input into your Requirements workflow efforts.

Your business model shows how your system fits into its environment and helps you to evolve a common understanding with your project stakeholders.

1.4.2 The Requirements Workflow

The purpose of the Requirements workflow, the topic of Chapter 3, is to engineer the requirements for your project. During the Inception phase, you will:

Identify the initial requirements. You need to identify the requirements for your software to provide sufficient information for scoping, estimating, and planning your project. To do this, you often need to identify your requirements to the 10–20% level — to the point where you understand at a high-level what your project needs to deliver, but may not fully understand the details. You are likely to develop several artifacts as part of your requirements model, including, but not limited to, an essential use-case model (Constantine & Lockwood, 1999; Ambler, 2001), an essential user interface prototype (Constantine & Lockwood, 1999; Ambler, 2001), a user-interface flow diagram (Ambler, 2001), and a *supplementary specification* (Kruchten, 2000). A supplementary specification is a "catch-all" artifact where you document business rules, constraints, and non-functional requirements. During the Elaboration phase, you will develop your requirements model to the 80% level and finalize it during the Construction phase.

Use cases are only a small part of your overall requirements model.

Develop a vision for the project. The vision document summarizes the high-level requirements for a project, including its behavioral requirements, its technical requirements, and the applicable constraints placed on the project.

Elicit stakeholder needs. A software project often has a wide range of stakeholders, including, but not limited to, its end users, senior management, and your operations and support staff. An important part of understanding the requirements of a system is eliciting the needs of its stakeholders so that the overall vision for the project may be negotiated.

Define a common vocabulary. A project glossary should be started during the Inception phase and evolved throughout the entire lifecycle. This glossary should include a consistent and defined collection of business terms applicable to your system. It is also common to include technical terms pertinent to your project — such as *essential modeling*, *use case*, *iteration*, and *Java* — that your project stakeholders need to be familiar with.

Define the scope of the project. An important part of requirements engineering is the definition of the boundary, the scope, of your system. Your goal is to define what requirements your project team intends to fulfill and what requirements it will not. For example, if you were developing a banking system, you may decide that the current version will support international transactions, although it will do so in a single currency only — leaving multi-currency support for a future release of the system.

1.4.3 The Analysis and Design Workflow

The purpose of the Analysis and Design workflow is to model your software. During the Inception phase, your modeling efforts are likely to focus on understanding the fundamental business and the requirements of your system — activities of the Business Modeling and Requirements workflows. It is often quite common to identify, and then model, a candidate architecture that is likely to meet your project's needs — an architecture that should be prototyped as part of your Implementation workflow efforts. By showing that your candidate architecture works, you help to show the technical feasibility of your approach, an important aspect of justifying your project to senior management as part of the Project Management workflow.

For projects that involve significant integration with existing systems, you may find that you need to invest resources in understanding the interfaces that you have with those systems — an effort often called *legacy analysis*, *external interface analysis*, or simply *data analysis*. Your context model, developed as part of your business modeling efforts, will indicate the external systems that yours will interact with. Once these external systems are identified, you may decide to begin work analyzing the interfaces. Ideally, you would merely need to obtain access to the existing documentation, although you will often find that little or no documentation exists and therefore, significant analysis must occur. This work can begin in the Inception phase and continue into the Construction phase. Legacy analysis techniques are described in *The Unified Process Elaboration Phase* (Ambler & Constantine, 2000a).

Analysis of interfaces to legacy systems is often a significant part of the modeling efforts for a project, an activity that often begins during the Inception phase.